TEACHING ACCESS CONTROL WITH DOMAIN TYPE ENFORCEMENT

Steve Carr, Jean Mayo
Department of Computer Science
Michigan Technological University
Houghton MI 49931-1295

ABSTRACT

Access control systems have become significantly more sophisticated in order to meet the security demands of Internet-based computing. Students must be trained formally to deal with the complexity of these modern systems, to ensure correctness and to encourage the use of sophisticated technology when appropriate. In this paper, we describe our application of the Domain Type Enforcement (DTE) access control model to enhance education in this area. We believe its use can significantly enhance education in the fundamentals of access control.

INTRODUCTION

A guiding principle for the design of trusted systems is the *principle of least privilege*, which states that a subject should be given only those privileges required for it to perform its intended function. When a server that runs as *root* is compromised, it should not be able to modify **any** system file. Access control systems are a fundamental mechanism for enforcing this principle. An access control system determines when a request by a subject (e.g., process) for access to an object (e.g., file) will be granted. The decision is made in accordance with the local access control policy, which defines what type of access (the rights or permissions) each subject is allowed to each object.

Application of this principle has required significant advances in operating system access control technology. Controls must be fine-grained, so that the rights of each individual subject can be distinguished from other subjects, and the nature of the access granted to each subject can be specified. Mandatory access control (MAC) is required. Mandatory control is not at the discretion of any single user. New models are helpful to simplify design and implementation of complex policies.

Both Windows and Linux have undergone significant changes to meet this requirement. The National Security Agency's SELinux project provided an impetus for advancement in the Linux community. The SELinux project [10,14] resulted in a version of Linux that supports MAC as well as non-traditional models of access control, including Type Enforcement (TE) [5], Multilevel Security (MLS) [2,3,4], and Role-based Access Control (RBAC) [12]. These enhancements were adopted into the RedHat Enterprise, Fedora Core, Gentoo, and Debian Linux distributions [9]. Windows attaches access-control lists (ACLs) to objects. Windows ACLs now comprise up to 30 different privileges for operations on about 15 different kinds of objects [6].

It is important that students receive formal education on the application of this new technology. These systems are very complex. A well-defined, strict TE policy on a Linux system can contain tens of thousands of rules [11]. On Windows systems, configuration errors have been found in the default installation of software from Adobe, AOL, Macromedia, Microsoft and others, that allow exploitable user-to-administrator and guest-to-any-user vulnerabilities [6]. Students must be prepared to manage this complexity and they must be encouraged to adopt sophisticated approaches to access control once they take up positions in industry.

This paper describes our application of the Domain Type Enforcement (DTE) [1] access control model to enhance education in this area. While DTE has existed for some time, most security textbooks do not cover it in detail, or simply ignore it. We believe that this is a missed opportunity. DTE is expressive enough to easily represent other common models, and is amenable to graphical depiction and analysis. DTE is a natural way to think about specifying access control policy and has an associated language, Domain Type Enforcement Language (DTEL), for precise specification of complex policies. We believe that DTE provides a framework for teaching access control models and DTEL allows students to specify policies in a manner that will increase student understanding. Finally, DTE is in use on real systems [7] and students gain useful practical knowledge.

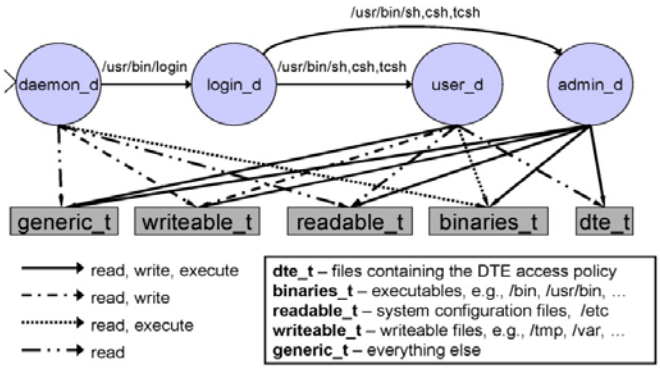

Figure 1: Graphical Depiction of Partial DTE Specification

EXPERIENCE

At Michigan Tech we have covered access control within courses in systems programming, operating systems, computer security, and system administration. Our course in computer security covers access control in depth. This course is at the senior level and has been offered for six years. We have found DTE/DTEL to be very effective for introducing students to fundamental concepts in access control.

Domain Type Enforcement

DTE [1] is an enhanced version of TE. Via DTE, a user essentially groups active entities into domains; groups passive entities into types; and specifies the kind of access each domain has to objects of a given type. A process may transition from one domain to another via execution of an entry point program. This transition may be either discretionary or automatic.

As an example, consider the problem of disallowing users to modify system binaries on a standard UNIX system. This could, for example, prevent introduction of a Trojan horse by a user process. Via DTE, one can put processes executed by users into one domain and processes executed by administrators into a different domain. Each of the two domains has different access to system files. Users may only execute system binaries, while administrators may both write and execute system binaries.

A graphical depiction of this solution is given in Figure 1. The first UNIX process, ***/sbin/init***, executes in domain ***daemon_d***. Any program executed from this domain creates a process in the

daemon_d domain, with one exception. Execution of the entry point program ***/usr/bin/login*** creates a process in the ***login_d*** domain. Execution of one of the shells ***/usr/bin/sh***, ***/usr/bin/csh***, or ***/usr/bin/tcsh*** creates a process in either the ***user_d*** or the ***admin_d*** domains (as determined by ***/usr/bin/login***). A process cannot transition out of either the ***user_d*** or the ***admin_d*** domain.

Files are collected into one of five types: ***generic_t***, ***write_t***, ***read_t***, ***bin_t***, or ***dte_***t. The ***read_t*** type contains the system configuration files, e.g., the password file. The ***write_t*** type contains the writeable files, e.g., log files and printer spools. The ***bin_t*** type contains the system binaries. The ***dte_t*** type contains the access policy. Finally, the ***generic_t*** type contains everything else.

Processes executing in the ***admin_d*** domain have unlimited access to all files in the system. Processes in the ***user_d*** domain may only read and execute the system binaries; may only read the DTE policy and system configuration files; may read and write ***write_t*** files; and may read, write, and execute ***generic_t*** files. Processes in the ***daemon_d*** domain have no rights to the access policy. Processes in this domain may read and write the ***write_t*** files, they may read and execute the system binaries, and they may only read the remaining files. Rights for the ***login_d*** domain are not shown in Figure 1.

Notice that the only transition from ***daemon_d*** is via execution of ***/usr/bin/login***. A daemon (started by ***init*** or one of its descendants) executing in this domain can never write the system binaries (and introduce a Trojan horse). Transition to another domain requires successful execution of ***/usr/bin/login***, which will require submission of valid credentials.

```
domain daemon_d =       (/sbin/init),
                        (crwd->write_t),
                        (rxd->bin_t),
                        (rd->generic_t,read_t),
                        (auto->login_d);

initial_domain=         daemon_d;

assign –r generic_t /;
assign –r read_t /etc;
assign –r –s bin_t /bin, /sbin;
```

```
domain login_d =        (/usr/bin/login),
                        (crwd->write_t),
                        (rd->generic_t, read_t),
                        setauth,
                        (exec->user_d,admin_d);

type dte_t, bin_t, write_t, generic_t, read_t;

assign –r write_t /usr/bar, /dev, /tmp;
assign –r –s dte_t /dte;
assign –r –s bin_t /usr/bin, /usr/sbin
```

Figure 2: DTE Specification of MAC for Standard UNIX Protections

Figure 2 depicts a portion of the specification. These components were selected to illustrate DTEL. A full DTEL specification for this solution is given in [15]. The ***type*** statement declares equivalence classes of data that are treated differently by the policy. Files are assigned to a given type via the ***assign*** statement. The ***-r*** flag to the ***assign*** statement indicates that all files rooted at the indicated directory be assigned the specified type; assignment statements appearing later in the specification override earlier assignments. The ***-s*** flag to the ***assign*** statement indicates that the type cannot be changed at run time, even if the file is replaced by a different file. The ***domain*** statement defines modes in which a process running in the domain may access objects of a given type. Access mode flags include: ***c***, to indicate the default type for created objects; ***d***, for directory traversal; and ***r***, ***w***, and ***x***, for read, write, and execute, respectively. The specification of Figure 2 indicates that the entry point program for the domain ***login_d*** is ***/usr/bin/login***. The transition from domain A to domain B is permitted only if the specification for domain A specifies one of the ***auto*** or ***exec*** access modes for domain B. The ***auto***

access mode specifies an automatic transition to the indicated domain upon execution of an entry point program. The ***exec*** mode indicates the transition is at the discretion of the executing process.

Benefits

DTE has proven to be a natural way for students to think about access control policy. We believe DTE provides the following benefits for teaching access control.

A Process-oriented Paradigm. Students typically have experience with Windows ACLs or the UNIX paradigm. Both these systems encourage students to view the problem of specifying access control policy from the perspective of the object, or file. One specifies which subjects may access that object. Application of the principle of least privilege follows more naturally when one views the problem from the perspective of a subject. The policy must specify exactly which objects a subject may access (and how the subject may access the object).

A Language for Policy Specification. DTEL provides a precise mechanism for specifying access control policies. This encourages students to think precisely about access control policy. It also facilitates assignments dealing with relatively complex access control problems.

Expressive. DTE can be used to generate specifications that implement other common access control models. For example, one can implement a simple MLS or RBAC policy via DTE. Due to space restrictions, we do not discuss how this can be done.

Graphical Depiction and Analysis. A DTE specification can be represented by a directed graph *G=(V,E)*. In a simple representation, *G* contains a node *d* for each domain and a node *t* for each object type. There is an edge from *d* to *d'* if a transition from *d* to *d'* is possible; an edge exists from *d* to *t* if *d* is allowed to access objects of type *t*. Analysis of the graph reveals characteristics of the policy [8]. For example, in the graph of Figure 1 the set of all objects accessible from processes executing in domain ***daemon_d*** is precisely the set of object nodes that are direct successors of the ***daemon_d*** node. The set of all domains that can be entered by a process in domain ***user_d*** is empty because no domain node is reachable from the ***user_d*** node.

Lessons

The following section presents classroom lessons that illustrate the benefits of using DTE for teaching access control. Currently exercises are paper based. However, we are investigating incorporation of automated tools that would allow us to assign problems with a level of complexity that more closely matches real systems, and to give a broader range of assignments to students.

Principle of Least Privilege. We introduce students to DTE/DTEL and the principle of least privilege by having them modify an existing specification in order to contain an individual program or service. An example is containment of an installation service that (only) reads and writes to the system binaries and reads and writes several log files. This is a straightforward specification in DTE. Execution of the service binary causes entry into a domain that allows only the specified access. This is not as straightforward in UNIX. If the service executes as root, it attains unnecessary privileges. If a special group is created for read and write access, then additional information about the access requirements of other processes is required to ensure that if the group access bits are used for this service, then the group neither restricts the access required by other processes nor grants unnecessary privilege. Students using group access bits to implement least privilege often make both of these mistakes. Thus,

in addition to the principle of least privilege and DTE/DTEL, this exercise helps students to understand the limitations on access control that arise from coarse granularity.

Mandatory versus Discretionary Controls. DTE is typically implemented as a mandatory control that overlays the standard UNIX discretionary controls. DTE helps students to understand the relationship between these two approaches. An example is specification of privileges for ***/usr/bin/newgrp***. This binary is typically SUID root, but may be executed by any user. If compromised on a system with traditional controls, this process may have access to all system files. Under DTE, this process can be contained to a subset of the system files. This illustrates the ability of mandatory controls to limit the damage that may be inflicted by an errant program and reinforces the importance of the principle of least privilege.

Virus Containment. DTE nicely supports a discussion of the role of access control in the containment of a virus. While students understand how a particular process becomes infected, they have more difficulty envisioning paths through which a virus has the potential to spread. A graphical representation of a DTE policy allows students to trace the exact sequence of accesses by which a virus might propagate. This again reinforces the principle of least privilege, and helps them to appreciate the limitations of access control. An example may require students to specify all files that can be accessed during an execution of ***/usr/bin/newgrp*** from within the ***user_d*** domain of the example specification. Even if this binary is executed as SUID root and then compromised, a student can see that it will never be able to overwrite the system binaries. At the same time, they see that there is no protection for the files that can be written from within this domain.

Multilevel Security. Students commonly have difficulty understanding the MLS model and its practical limitations. We have them implement an abstract MLS scheme using DTEL. The scheme might solve a real problem, such as controlling access to the files for an engineering firm responsible for a sensitive design, e.g., of a new military jet. This concrete specification for a UNIX system requires students to think through the practical implications of the MLS scheme. It also serves as a basis for discussing the role of tranquility within the Bell-LaPadula model, which is difficult for students to fully understand otherwise.

RELATED WORK

Schweitzer, Collins, and Baird developed a visualization system to enable active learning about the Harrison, Ruzzo, Ullman and Take-Grant models of access control [13]. Their work focuses on understanding operation of only these two models.

Hallyn and Kearns developed DTEEdit and DTEView for graphical analysis of DTE specifications [8]. Their tool takes a policy file as input and produces a plugin for a Perl/Tk graphical analysis tool. At the time of the writing of this paper, the distribution URL was stale, so the full functionality of their tool could not be determined. There has been no work on the application of these tools to education. We are investigating classroom use of this tool.

CONCLUSIONS

Access control systems have evolved substantially to meet the security demands of Internet-based computing. It is important that students receive formal education on these new systems, so that

they manage the complexity correctly and to encourage them to adopt these more sophisticated systems when it is appropriate.

We believe Domain Type Enforcement (DTE) is a useful model around which to structure access control education. DTE encourages application of the principle of least privilege, it has an associated specification language, it is expressive enough to represent other common access control models, and it is amenable to graphical depiction and analysis. We are currently investigating the use of software tools, like DTEEdit and DTEView, to enhance the use of DTE in the classroom.

REFERENCES

[1] Badger, M. L., Sterne, D. F., Sherman, D. L., Walker, K. M., A domain and type enforcement UNIX prototype, *Computing Systems*, 9(1), 47-83, 1996.

[2] Bell, D., La Padula, L., Secure computer systems: Mathematical foundations, Technical Report MTR-2547, Vol 1, MITRE Corp., Bedford, MA, Nov. 1973.

[3] Biba, K. J., Integrity considerations for secure computer systems, MTR-3153, Rev. 1, MITRE Corporation, Bedford, MA, April 1977.

[4] Bidgoli, H., editor, *Handbook of Information Security, Volume 3: Threats, Vulnerabilities, Prevention Detection and Management*, John Wiley & Sons, 2005.

[5] Boebert, W. E., Kain, R. Y., A practical alternative to hierarchical integrity policies, *Proc. National Computer Security Conference*, page 18, Oct. 1985.

[6] Govindavajhala, S., Appel, A. W., Windows access control demystified, Technical Report TR-744-06, Princeton University, Princeton, NJ, Jan. 2006.

[7] Hallyn, S., Kearns, P., Domain and type enforcement for linux, *Proceedings of the 4th Annual Showcase and Conference* (LINUX-00), pages 247-260, Oct. 2000.

[8] Hallyn, S., Kearns, P., Tools to administer domain and type enforcement, *Proceedings of the Fifteenth Systems Administration Conference* (LISA XV), page 151, Dec. 2001.

[9] Mayer, F., Macmillan, K., Caplan, D*., SELinux by Example: Understanding Security Enhanced Linux*, Prentice Hall, 2007.

[10] McCarty, B., *SELINUX: NSA's open source Security Enhanced Linux*, O'Reilly & Associates, Inc., 2005.

[11] Red Hat, Inc., Red hat enterprise linux 4: Red hat 9 guide., http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/9-gui%de/rhlcommon-section-0104.html, 2005 (accessed January 7, 2010).

[12] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., Youman, C. E., Role-based access control models, *IEEE Computer*, 20(2), 38-47, 1996.

[13] Schweitzer, D., Collins, M., Baird, L., A visual approach to teaching formal access models in security, In *Proceedings of the 11th Colloquium for Information Systems Security Education*, pages 69-75, June 2007.

[14] Smalley, S., Vance, C., Salamon, W., Implementing SELinux as a Linux security module, Report #01-043, NAI Labs, Dec. 2001, Revised May 2002.

[15] Walker, K. M., Sterne, D. F., Badger, M. L., Petkac, M. J., Sherman, D. L., Oostendorp, K. A., Confining root programs with domain and type enforcement, In *Proceedings of the Sixth USENIX Security Symposium*, pages 21 -36, July 1996.